एक □□-संचालित कहानी बॉट बनाएं

यह ब्लॉग पोस्ट □□□□□□□-4 की सहायता से लिखा गया है।

विषयसूची

परिचय

यह ब्लॉग पोस्ट एक □□-संचालित स्टोरी बॉट एप्लिकेशन की आर्किटेक्चर और कार्यान्वयन के लिए एक व्यापक गाइड प्रदान करती है। इस प्रोजेक्ट में वेब इंटरफेस का उपयोग करके व्यक्तिगत कहानियाँ उत्पन्न करना शामिल है। हम विकास के लिए □□□□□□, □□□□□, और

□□□□□ का उपयोग करते हैं और □□□ पर डिप्लॉय करते हैं। इसके अतिरिक्त, हम मॉनिटरिंग के लिए □□□□□□□□□□ और लॉग प्रबंधन के लिए □□□□□□□□□□□□□□, □□□□□□□, और □□□□□□□□□ का उपयोग करते हैं। □□□ प्रबंधन □□□□□□□□ और □□□□□□□□□□ के माध्यम से किया जाता है, और □□□ प्रमाणपत्र और अनुरोध हेडर प्रबंधन के लिए □□□□□ को गेटवे के रूप में उपयोग किया जाता है।

प्रोजेक्ट आर्किटेक्चर

बैकएंड प्रोजेक्ट का बैकएंड □□□□□ का उपयोग करके बनाया गया है, जो □□□□□□ में एक हल्का □□□□ वेब एप्लिकेशन फ्रेमवर्क है। बैकएंड □□□ अनुरोधों को संभालता है, डेटाबेस का प्रबंधन करता है, एप्लिकेशन गतिविधियों को लॉग करता है, और मॉनिटरिंग के लिए □□□□□□□□□□ के साथ एकीकृत होता है।

यहां बैकएंड घटकों का विवरण दिया गया है:

1. □□□□□ एप्लिकेशन सेटअप:
   - □ □□□□□ ऐप को प्रारंभ किया गया है और इसे विभिन्न एक्सटेंशन जैसे □□□□□-□□□□ (क्रॉस-ओरिजिन रिसोर्स शेयरिंग को संभालने के लिए) और □□□□□-□□□□□□□ (डेटाबेस माइग्रेशन को प्रबंधित करने के लिए) का उपयोग करने के लिए कॉन्फ़िगर किया गया है।
   - □ एप्लिकेशन रूट्स को प्रारंभ किया गया है, और क्रॉस-ओरिजिन अनुरोधों की अनुमति देने के लिए □□□□ सक्षम किया गया है।
   - □ डेटाबेस को डिफ़ॉल्ट कॉन्फ़िगरेशन के साथ प्रारंभ किया गया है, और □□□□□□□□ के लिए लॉग एंट्री को फॉर्मेट करने के लिए एक कस्टम लॉगर सेट किया गया है।

```
    from flask import Flask
    from flask_cors import CORS
    from .routes import initialize_routes
    from .models import db, insert_default_config
    from flask_migrate import Migrate
    import logging
    from logging.handlers import RotatingFileHandler
    from prometheus_client import Counter, generate_latest, Gauge

app = Flask(__name__)
app.config.from_object('api.config.BaseConfig')

db.init_app(app)
initialize_routes(app)
CORS(app)
migrate = Migrate(app, db)
```

2. लॉगिंग और मॉनिटरिंग:

    - एप्लिकेशन लॉग फ़ाइलों को प्रबंधित करने के लिए □□□□□□□□□□□□□□□□□□□□ का उपयोग करता है और लॉग्स को एक कस्टम फॉर्मेटर का उपयोग करके फॉर्मेट करता है।
    - एप्लिकेशन में □□□□□□□□□□ मेट्रिक्स को एकीकृत किया गया है ताकि अनुरोधों की संख्या और विलंबता को ट्रैक किया जा सके।

```
REQUEST_COUNT = Counter('flask_app_request_count', 'Flask             ', ['method', 'endpoint', 'http_s
REQUEST_LATENCY = Gauge('flask_app_request_latency_seconds', '       ', ['method', 'endpoint'])

def setup_loggers():
    logstash_handler = RotatingFileHandler('app.log', maxBytes=100000000, backupCount=1)
    logstash_handler.setLevel(logging.DEBUG)
    logstash_formatter = CustomLogstashFormatter()
    logstash_handler.setFormatter(logstash_formatter)
```

यह फ़ंक्शन `setup_loggers` लॉगर्स को सेटअप करने के लिए है। इसमें `RotatingFileHandler` का उपयोग करके एक लॉग फ़ाइल `app.log` बनाई जाती है, जिसका अधिकतम आकार 100,000,000 बाइट्स होता है और केवल 1 बैकअप फ़ाइल रखी जाती है। लॉग लेवल को `DEBUG` पर सेट किया जाता है और `CustomLogstashFormatter` का उपयोग करके लॉग फ़ॉर्मेटर सेट किया जाता है।

```
root_logger = logging.getLogger()
root_logger.setLevel(logging.DEBUG)
root_logger.addHandler(logstash_handler)
```

यह कोड □□□□□□□ में □□□□□□□□ को □□□□□□□□□□ करने के लिए है। `root_logger` नाम का एक □□□□□□□ बनाया गया है, जिसका □□□ □□□□□□ `DEBUG` पर सेट किया गया है। इसके बाद, `logstash_handler` नाम का एक □□□□□□□□ जोड़ा गया है, जो □□□□□ को □□□□□□□□□□ में भेजने के लिए उपयोग किया जाता है।

```
app.logger.addHandler(logstash_handler)
werkzeug_logger = logging.getLogger('werkzeug')
werkzeug_logger.setLevel(logging.DEBUG)
werkzeug_logger.addHandler(logstash_handler)

setup_loggers()
```

3. अनुरोध प्रबंधन:

    - एप्लिकेशन प्रत्येक अनुरोध से पहले और बाद में मेट्रिक्स को कैप्चर करता है, और अनुरोध प्रवाह को ट्रैक करने के लिए एक ट्रेस □□ जनरेट करता है।

```
    def generate_trace_id(length=4):
        characters = string.ascii_letters + string.digits
        return ''.join(random.choice(characters) for _ in range(length))

@app.before_request
def before_request():
    request.start_time = time.time()
    trace_id = request.headers.get('X-Trace-Id', generate_trace_id())
    g.trace_id = trace_id
```

यह कोड □□□□□ एप्लिकेशन में एक `before_request` फंक्शन को परिभाषित करता है। यह फंक्शन हर □□□□ रिक्वेस्ट से पहले चलता है। इसमें:

1. `request.start_time` को वर्तमान समय (`time.time()`) के साथ सेट किया जाता है।
2. `X-Trace-Id` हेडर से ट्रेस आईडी प्राप्त की जाती है। अगर यह हेडर उपलब्ध नहीं है, तो `generate_trace_id()` फंक्शन का उपयोग करके एक नई ट्रेस आईडी जेनरेट की जाती है।
3. इस ट्रेस आईडी को `g.trace_id` में स्टोर किया जाता है, जो □□□□□ के `g` ऑब्जेक्ट का उपयोग करता है और यह रिक्वेस्ट के दौरान ग्लोबल रूप से उपलब्ध होता है।

```
    @app.after_request
    def after_request(response):
        response.headers['X-Trace-Id'] = g.trace_id
        request_latency = time.time() - getattr(request, 'start_time', time.time())
        REQUEST_COUNT.labels(method=request.method, endpoint=request.path, http_status=response.status_
        REQUEST_LATENCY.labels(method=request.method, endpoint=request.path).set(request_latency)
        return response
```

फ्रंटएंड प्रोजेक्ट का फ्रंटएंड □□□□□ का उपयोग करके बनाया गया है, जो यूजर इंटरफेस बनाने के लिए एक □□□□□□□□□□ लाइब्रेरी है। यह बैकएंड □□□ के साथ इंटरैक्ट करता है ताकि कहानी प्रॉम्प्ट्स को प्रबंधित किया जा सके और यह व्यक्तिगत कहानियों को जनरेट और प्रबंधित करने के लिए एक इंटरैक्टिव यूजर इंटरफेस प्रदान करता है।

1. □□□□□ □□□□□□□□□□:
    - □ मुख्य कंपोनेंट स्टोरी प्रॉम्प्ट्स के लिए यूजर इनपुट को संभालता है और इन कहानियों को प्रबंधित करने के लिए बैकएंड □□□ के साथ इंटरैक्ट करता है।

```
    import React, { useState, useEffect } from 'react';
    import { ToastContainer, toast } from 'react-toastify';
```

```
    import 'react-toastify/dist/ReactToastify.css';
    import { apiFetch } from './api';
    import './App.css';

    function App() {
      const [prompts, setPrompts] = useState([]);
      const [newPrompt, setNewPrompt] = useState('');
      const [isLoading, setIsLoading] = useState(false);

useEffect(() => {
  fetchPrompts();
}, []);

const fetchPrompts = async () => {
  setIsLoading(true);
  try {
    const response = await apiFetch('prompts');
    if (response.ok) {
      const data = await response.json();
      setPrompts(data);
    } else {
      toast.error('               ');
    }
  } catch (error) {
    toast.error('                    ');
  } finally {
    setIsLoading(false);
  }
};

const addPrompt = async () => {
  if (!newPrompt) {
    toast.warn('Prompt content cannot be empty');
    return;
  }
  setIsLoading(true);
  try {
```

```
    const response = await apiFetch('prompts', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ content: newPrompt }),
    });
    if (response.ok) {
      fetchPrompts();
      setNewPrompt('');
      toast.success('Prompt added successfully');
    } else {
      toast.error('Failed to add prompt');
    }
  } catch (error) {
    toast.error('An error occurred while adding the prompt');
  } finally {
    setIsLoading(false);
  }
};
```

हिंदी अनुवाद:

```
const addPrompt = async () => {
  if (!newPrompt) {
    toast.warn('                    ');
    return;
  }
  setIsLoading(true);
  try {
    const response = await apiFetch('prompts', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ content: newPrompt }),
    });
```

```
    if (response.ok) {
      fetchPrompts();
      setNewPrompt('');
      toast.success('               ');
    } else {
      toast.error('           ');
    }
  } catch (error) {
    toast.error('                ');
  } finally {
    setIsLoading(false);
  }
};

  const deletePrompt = async (promptId) => {
    setIsLoading(true);
    try {
      const response = await apiFetch(`prompts/${promptId}`, {
        method: 'DELETE',
      });
      if (response.ok) {
        fetchPrompts();
        toast.success('                 ');
      } else {
        toast.error('           ');
      }
    } catch (error) {
      toast.error('                ');
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="app">
      <h1>AI-         </h1>
      <div>
```

```
        <input
          type="text"
          value={newPrompt}
          onChange={(e) => setNewPrompt(e.target.value)}
          placeholder="      "
        />
        <button onClick={addPrompt} disabled={isLoading}>      </button>
      </div>
      {isLoading ? (
        <p>        ...</p>
      ) : (
        <ul>
          {prompts.map((prompt) => (
            <li key={prompt.id}>
              {prompt.content}
              <button onClick={() => deletePrompt(prompt.id)}>  </button>
            </li>
          ))}
        </ul>
      )}
      <ToastContainer />
    </div>
  );
}

    export default App;
    ```

     JavaScript   React   `App`   ,

2. API   :
    -    ,   API   fetch

```javascript
    export const apiFetch = (endpoint, options) => {
      return fetch(`https://api.yourdomain.com/${endpoint}`, options);
    };
```

```

###

AWS , DNS GoDaddy Cloudflare SSL Nginx

1. :
- , Fabric

```python
from fabric import task
from fabric import Connection

    server_dir = '/home/project/server'
    web_tmp_dir = '/home/project/server/tmp'

@task
def prepare_remote_dirs(c):
    if not c.run(f'test -d {server_dir}', warn=True).ok:
        c.sudo(f'mkdir -p {server_dir}')
        c.sudo(f'chmod -R 755 {server_dir}')
        c.sudo(f'chmod -R 777 {web_tmp_dir}')
        c.sudo(f'chown -R ec2-user:ec2-user {server_dir}')

@task
def deploy(c, install='false'):
    prepare_remote_dirs(c)
    pem_file = './aws-keypair.pem'
    rsync_command = (f'rsync -avz --exclude="api/db.sqlite3" '
                     f'-e "ssh -i {pem_file}" --rsync-path="sudo rsync" '
                     f'{tmp_dir}/ {c.user}@{c.host}:{server_dir}')
    c.local(rsync_command)
    c.sudo(f'chown -R ec2-user:ec2-user {server_dir}')
```

2. □□□□□□□□□□□□□□□ कॉन्फ़िगरेशन:

□ □□□□□□□□□□□□□□□ सेटअप में क्लस्टर, नोड, और नेटवर्क सेटिंग्स के लिए कॉन्फ़िगरेशन शामिल हैं।
```

```
cluster.name: my-application
node.name: node-1
path.data: /var/lib/elasticsearch
path.logs: /var/log/elasticsearch
network.host: 0.0.0.0
http.port: 9200
discovery.seed_hosts: ["127.0.0.1"]
cluster.initial_master_nodes: ["node-1"]
```

3. □□□□□□□ कॉन्फ़िगरेशन:

   - □ □□□□□□□ सेटअप में सर्वर और □□□□□□□□□□□□□□□ होस्ट्स के लिए कॉन्फ़िगरेशन शामिल हैं।

```
server.port: 5601
server.host: "0.0.0.0"
elasticsearch.hosts: ["http://localhost:9200"]
```

4. □□□□□□□□□ कॉन्फ़िगरेशन:

   - □ □□□□□□□□□ को लॉग फ़ाइलों को पढ़ने, उन्हें पार्स करने और पार्स किए गए लॉग्स को □□□□□□□□□□□□□□□ में आउटपुट करने के लिए कॉन्फ़िगर किया गया है।

```
input {
  file {
    path => "/home/project/server/app.log"
    start_position => "beginning"
    sincedb_path => "/dev/null"
  }
}

filter {
  json {
    source => "message"
  }
}
```

(यह कोड ब्लॉक है, इसे अनुवादित नहीं किया जाना चाहिए।)

```
output {
  elasticsearch {
    hosts => ["http://localhost:9200"]
    index => "flask-logs-%{+YYYY.MM.dd}"
  }
}
```

□□□□□ **कॉन्फ़िगरेशन और** □□□'□ □□□□□□□ □□□ **प्रमाणपत्र**

सुरक्षित संचार सुनिश्चित करने के लिए, हम □□□□□ को रिवर्स प्रॉक्सी के रूप में और □□□'□ □□□□□□□ को □□□ प्रमाणपत्रों के लिए उपयोग करते हैं। नीचे □□□□ से □□□□□ रीडायरेक्शन और □□□ प्रमाणपत्रों को सेटअप करने के लिए □□□□□ कॉन्फ़िगरेशन दिया गया है।

1. अनुमत मूल (□□□□□□□ □□□□□□□) को संभालने के लिए एक मैप (□□□) परिभाषित करें:

    ```
    map $http_origin $cors_origin {
        default "https://example.com";
        "http://localhost:3000" "http://localhost:3000";
        "https://example.com" "https://example.com";
        "https://www.example.com" "https://www.example.com";
    }
    ```

2. HTTP HTTPS :

```nginx
    server {
        listen 80;
        server_name example.com api.example.com;

    return 301 https://$host$request_uri;
}
```

3. example.com के लिए मुख्य साइट कॉन्फ़िगरेशन:

```
server {
    listen 443 ssl;
    server_name example.com;

ssl_certificate /etc/letsencrypt/live/example.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/example.com/privkey.pem;

    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";

    root /home/project/web;
    index index.html index.htm index.php default.html default.htm default.php;
```

यह कोड एक वेब सर्वर कॉन्फ़िगरेशन का हिस्सा है। यहां `root` डायरेक्टिव वेब सर्वर को बताता है कि वेबसाइट की फाइलें कहां स्थित हैं, और `index` डायरेक्टिव यह निर्धारित करता है कि जब कोई डायरेक्टरी अनुरोध की जाती है, तो कौन सी फाइलें डिफ़ॉल्ट रूप से लोड की जाएंगी।

```
location / {
    try_files $uri $uri/ =404;
}
```

यह कोड □□□□□ कॉन्फ़िगरेशन का एक हिस्सा है। इसे हिंदी में समझाएं:

- `location / { ... }`: यह □□□□□ में एक लोकेशन ब्लॉक है जो रूट □□□ (`/`) के लिए कॉन्फ़िगरेशन को परिभाषित करता है।
- `try_files $uri $uri/ =404;`: यह निर्देश □□□□□ को फ़ाइलों को खोजने का प्रयास करने के लिए कहता है:
  - `$uri`: यह अनुरोधित □□□ के अनुरूप फ़ाइल को खोजने का प्रयास करता है।
  - `$uri/`: यदि फ़ाइल नहीं मिलती है, तो यह एक डायरेक्टरी के रूप में खोजने का प्रयास करता है।
  - `=404`: यदि दोनों प्रयास विफल होते हैं, तो □□□□□ एक 404 त्रुटि (पेज नहीं मिला) लौटाता है।

इसका मतलब है कि यदि अनुरोधित □□□ के अनुरूप कोई फ़ाइल या डायरेक्टरी नहीं मिलती है, तो □□□□□ एक 404 त्रुटि पेज दिखाएगा।

```
    location ~ .*\.(gif|jpg|jpeg|png|bmp|swf)$ {
        expires 30d;
    }
```

यह □□□□□ कॉन्फ़िगरेशन ब्लॉक सभी □□□, □□□, □□□□, □□□, □□□, और □□□ फ़ाइलों के लिए कैशिंग नियम निर्धारित करता है। इन फ़ाइलों को ब्राउज़र में 30 दिनों तक कैश किया जाएगा।

```
    location ~ .*\.(js|css)?$ {
        expires 12h;
    }

    error_page 404 /index.html;
    }
    ```

4. `api.example.com`   API   :

```nginx
    server {
        listen 443 ssl;
        server_name api.example.com;

    ssl_certificate /etc/letsencrypt/live/example.com-0001/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/example.com-0001/privkey.pem;

        ssl_protocols TLSv1.2 TLSv1.3;
        ssl_prefer_server_ciphers on;
        ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH";

location / {
    #        Access-Control
    more_clear_headers 'Access-Control-Allow-Origin';
}

# CORS
if ($request_method = 'OPTIONS') {
    add_header 'Access-Control-Allow-Origin' $cors_origin;
    add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE';
    add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept, Authorization, X-Client-In
    add_header 'Access-Control-Max-Age' 3600;
    return 204;
}
```

```
add_header 'Access-Control-Allow-Origin' $cors_origin always;
add_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS, PUT, DELETE' always;
add_header 'Access-Control-Allow-Headers' 'Origin, Content-Type, Accept, Authorization, X-Client-Info, X

nginx             proxy_pass http://127.0.0.1:5000/;             proxy_set_header Host
$host;             proxy_set_header X-Real-IP $remote_addr;             proxy_set_header
X-Forwarded-For $proxy_add_x_forwarded_for;             proxy_set_header X-Forwarded-Proto
$scheme;             proxy_connect_timeout 600s;             proxy_send_timeout 600s;
proxy_read_timeout 600s;             send_timeout 600s;         }     }
```

निष्कर्ष

यह प्रोजेक्ट एक □□-संचालित स्टोरी बॉट एप्लिकेशन के लिए एक मजबूत आर्किटेक्चर प्रदर्शित करता है, जो आधुनिक वेब विकास प्रथाओं और उपकरणों का उपयोग करता है। बैकएंड □□□□□ के साथ बनाया गया है, जो लॉगिंग और मॉनिटरिंग के लिए विभिन्न सेवाओं के साथ कुशल अनुरोध प्रबंधन और एकीकरण सुनिश्चित करता है। फ्रंटएंड, □□□□□ के साथ बनाया गया है, जो स्टोरी प्रॉम्प्ट्स को प्रबंधित करने के लिए एक इंटरैक्टिव यूजर इंटरफेस प्रदान करता है। □□□ को डिप्लॉयमेंट के लिए, □□□□□ को सुरक्षित संचार के लिए, और □□□ स्टैक को लॉग प्रबंधन के लिए उपयोग करके, हम स्केलेबिलिटी, विश्वसनीयता और रखरखाव को सुनिश्चित करते हैं। यह व्यापक सेटअप एक सहज यूजर अनुभव प्रदान करने के लिए अत्याधुनिक तकनीकों को जोड़ने की शक्ति को प्रदर्शित करता है।